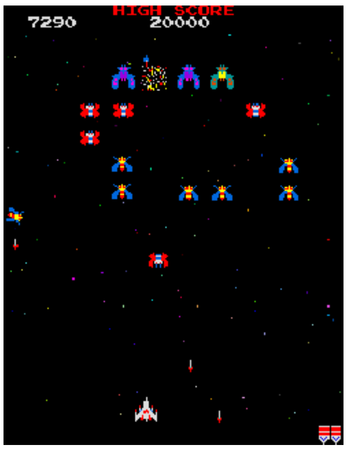
Concept: Star Destroyer

I started off by developing the Space Invader type game prototype given to us within the examples. From when I first played the prototype the idea of creating a type of game like retro shoot em ups like Galaga or Asteroids. I started by looking at the original gameplay of both games to draw inspiration from them.

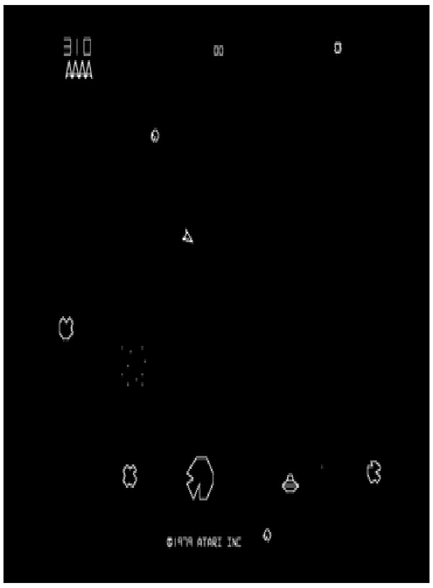
Galaga



Galaga (1981, Bandai Namco) is one of the first shoot em up games designed that's become a cult classic. The objective of the game is to gather as many points possible by shooting enemies. The player controls a spaceship that can move left to right along the bottom of the screen. The enemies fly in select formations, starting at the top of the screen and slowly gliding towards the bottom of the screen (and the player) whilst firing projectiles at the player and trying to smash into them. The gameplay ends either when the players 3 lives have been exhausted, or all enemies have been destroyed by the player.

I took inspiration from this game in the form of gameplay elements such as multiple player lives, player score and enemy projectiles.

Asteroids

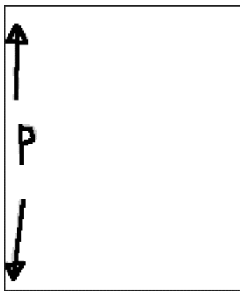


Asteroids (1979, Atari) is an arcade space shooter that was one of the first major hits of the 'golden age of arcade games'. The objective of the game is to destroy Asteroids and Saucers. The player controls a triangular ship that can fly, and shoot straight forward. Once the ship starts moving in one direction, it will continue in that direction for a time without player interference. The game ends when the players 3 lives have been exhausted.

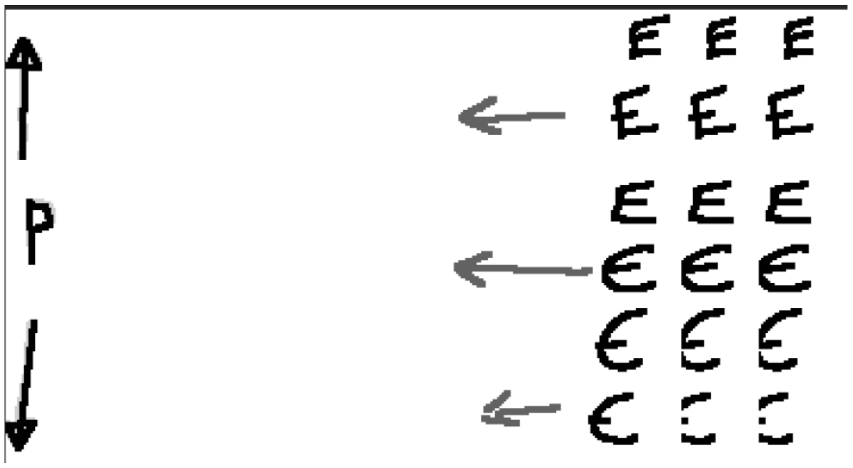
I took inspiration from this game within the form of the scoring, multiple player lives and the density of the enemy projectiles which will increase the difficulty of the game and hopefully the enjoyability of it.

Game Concept

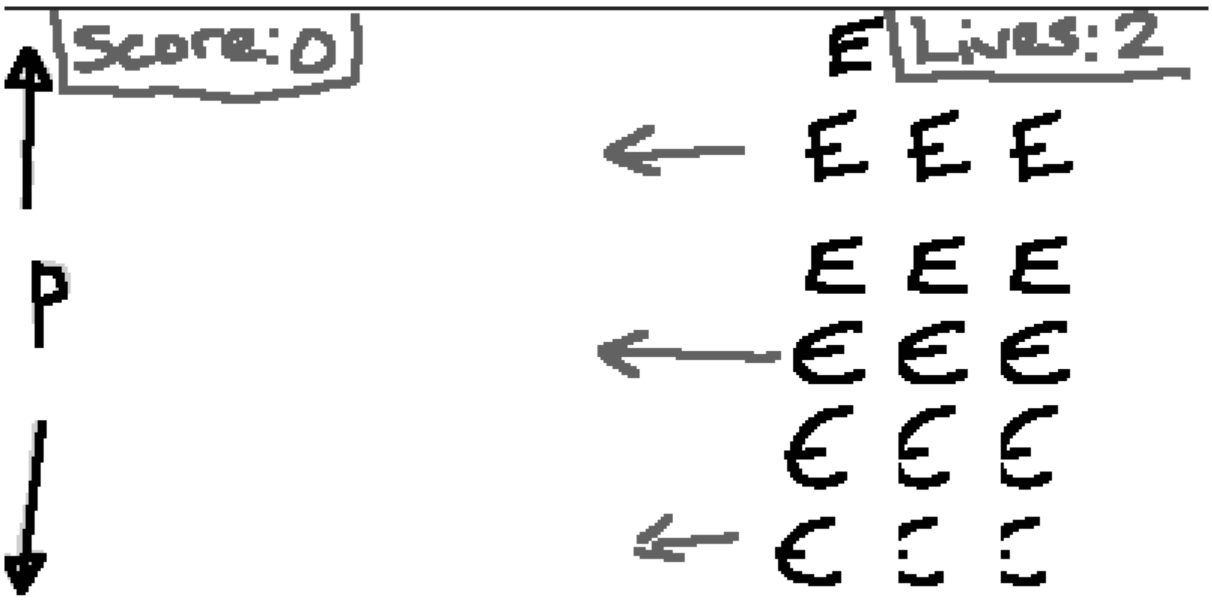
For my game I came up with the concept of having a side scrolling type game that used gameplay elements usually found within retro shoot em up games, hopefully with the same type of difficulty usually found within those type of games. As I said, my game would be a side scrolling type game where the player takes control of a Star Destroyer that can move up or down on one side of the screen:



The enemies would then spawn on the other side of the screen and slowly glide towards the player whilst shooting projectiles and trying to crash into the player. The goal of the enemies is to exhaust the players 3 lives.

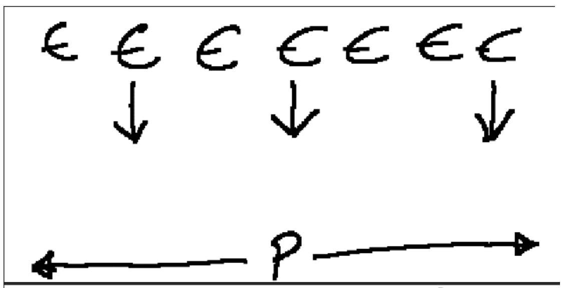


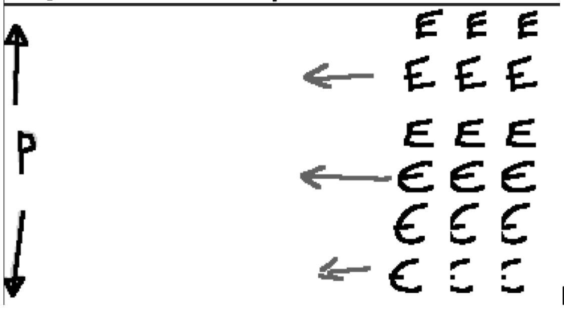
The player can retaliate at the enemies by shooting at them, if the player hits the enemy then the enemy would be destroyed, the player would gain an amount of score. The score will be displayed on the top corner next to the high score. This would change if the score within the game running is more than the existing high score. The players lives will also be shown at the other corner of the screen.



Development Process

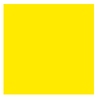
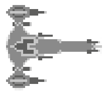
I started the development of my game by changing the placement of the players and the enemies and the movement of each one respectively from the bottom and top of the screen to either side of the screen:


Original

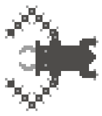

Revised Placement and movement

After establishing this block, I could easily build the rest of my game on top of this. I then proceeded to completely overhaul all the graphics within the game. Such as:

Player:

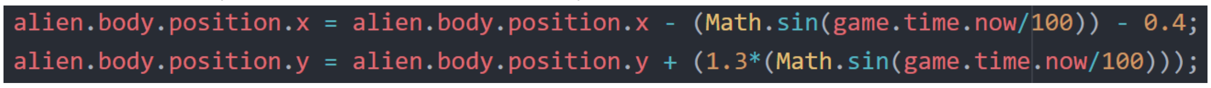
 to 

Enemy:

to 

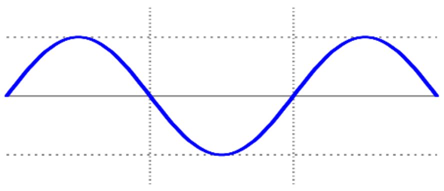
And so forth. Once I had completely overhauled the graphics I worked on developing the sounds. I did this by recording certain sounds in the outside world and heavily editing them, for example the explosion sound was that of a book being dropped on the floor.

After developing and implementing the sounds, I worked on making the enemy movement more interesting by making them move in a wave like formation instead of just straightforward. I started working on this by finding how to implement a Sin wave within JavaScript. I found you could do this by using an inbuilt function called Math.Sin(), which, when given a parameter will return a sine of a number. I then implemented this into the enemy movement function:

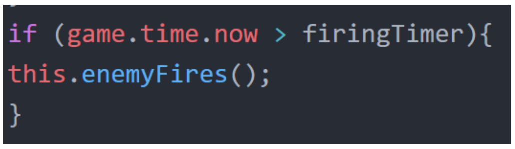


```
alien.body.position.x = alien.body.position.x - (Math.sin(game.time.now/100)) - 0.4;
alien.body.position.y = alien.body.position.y + (1.3*(Math.sin(game.time.now/100)));
```

This made the enemies move in a sine wave like this:

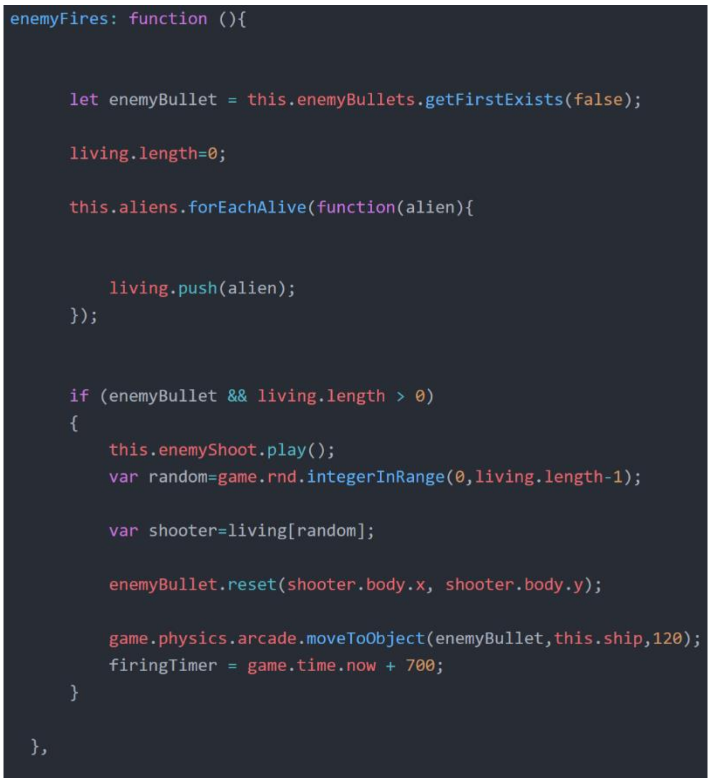

This made the enemy movement seem more organic and interesting then just having them walk straight to the player. Which I felt added to the overall game

feel. After implementing this function, I made it so that the enemies shot back at the player, adding somewhat of a danger to the gameplay.



```
if (game.time.now > firingTimer){
this.enemyFires();
}
```

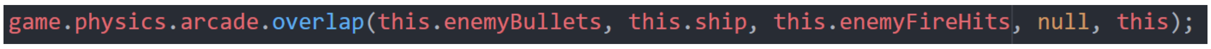
I started off by adding an if statement to the update function within the main game state, which when a given number was more than the game time, would call a function called “enemyFires”.



```
enemyFires: function (){


    let enemyBullet = this.enemyBullets.getFirstExists(false);

    living.length=0;

    this.aliens.forEachAlive(function(alien){


        living.push(alien);
    });


    if (enemyBullet && living.length > 0)
    {
        this.enemyShoot.play();
        var random=game.rnd.integerInRange(0,living.length-1);

        var shooter=living[random];

        enemyBullet.reset(shooter.body.x, shooter.body.y);

        game.physics.arcade.moveToObject(enemyBullet,this.ship,120);
        firingTimer = game.time.now + 700;
    }

  },
```

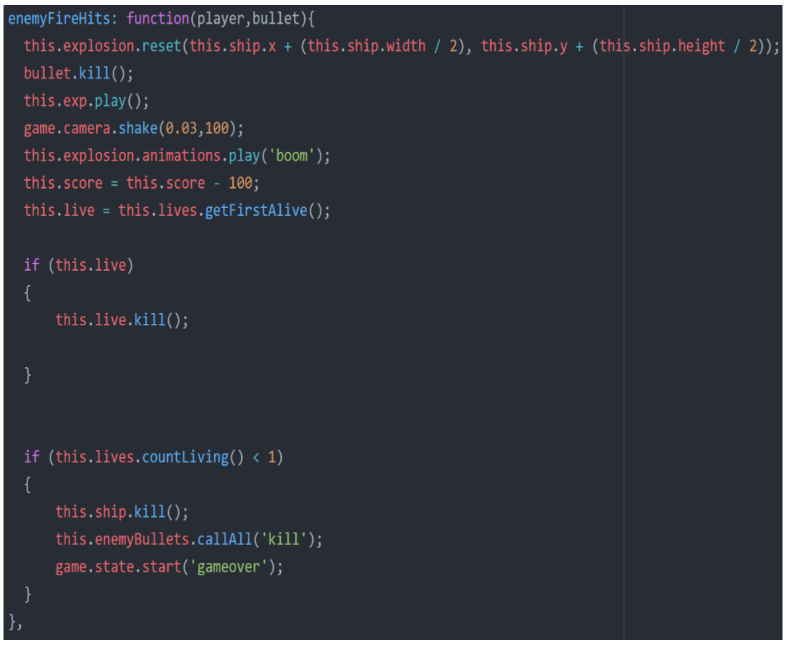
This function first gets all the available enemy projectiles, then proceeds to check for the remaining alive enemies, this is to stop projectiles just appearing from thin air, which would confuse the player. The function then checks if there’s available projectiles and whether there are any aliens alive, if so, the sound for the enemy firing plays, then selects a random alive alien out of the list of alive aliens. Then makes the sprite of the bullet appear before the enemy and then proceeds to make the bullet towards the players current position at the time of the bullet being fired. The function then updates the firing timer, this stops the aliens firing too many shots at the player at a time.

After doing this I had to create the collision for the enemy bullet and the player, which I simply had to do in the update function:



```
game.physics.arcade.overlap(this.enemyBullets, this.ship, this.enemyFireHits, null, this);
```

this code checks if the two sprites overlap, if so, then the function “enemyFireHits” is called:

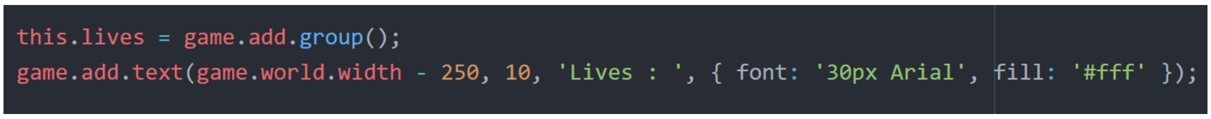


```
enemyFireHits: function(player,bullet){
  this.explosion.reset(this.ship.x + (this.ship.width / 2), this.ship.y + (this.ship.height / 2));
  bullet.kill();
  this.exp.play();
  game.camera.shake(0.03,100);
  this.explosion.animations.play('boom');
  this.score = this.score - 100;
  this.live = this.lives.getFirstAlive();

  if (this.live)
  {
      this.live.kill();

  }


  if (this.lives.countLiving() < 1)
  {
      this.ship.kill();
      this.enemyBullets.callAll('kill');
      game.state.start('gameover');
  }
},
```

This function plays the animation of the explosion sprite sheet at the player position, kills the bullet and plays the sound for the explosion. The game camera also shakes to create feeling within the 2d game. If the player is hit then 100 points are deducted from the score, the players life is then also deducted. If the player has no lives left, then the “game Over” state is called, and the game is over.

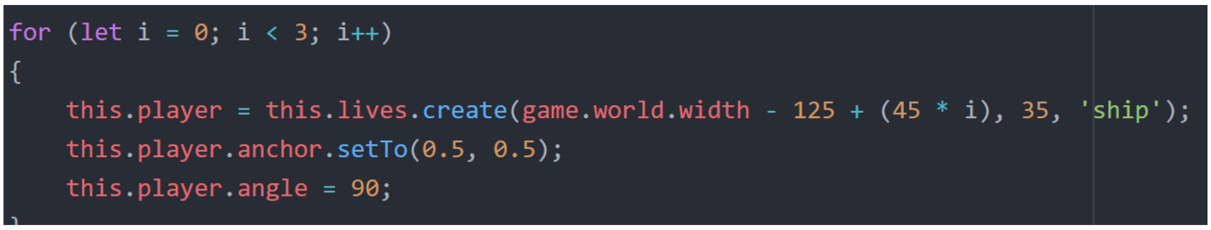
Player Lives:

Before I came up with the idea, the player only had one life that was easily taken by the enemies, so this did not feel too fair for the player. Therefore, I added multiple lives for the player, giving them multiple chances at winning the game.



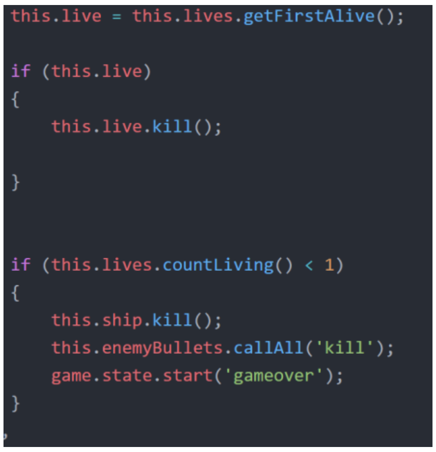
```
this.lives = game.add.group();
game.add.text(game.world.width - 250, 10, 'Lives : ', { font: '30px Arial', fill: '#fff' });
```

This code creates a group that will hold the players lives and also creates a line of the text on the players screen displaying the amount of lives the player has.



```
for (let i = 0; i < 3; i++)
{
    this.player = this.lives.create(game.world.width - 125 + (45 * i), 35, 'ship');
    this.player.anchor.setTo(0.5, 0.5);
    this.player.angle = 90;
}
```

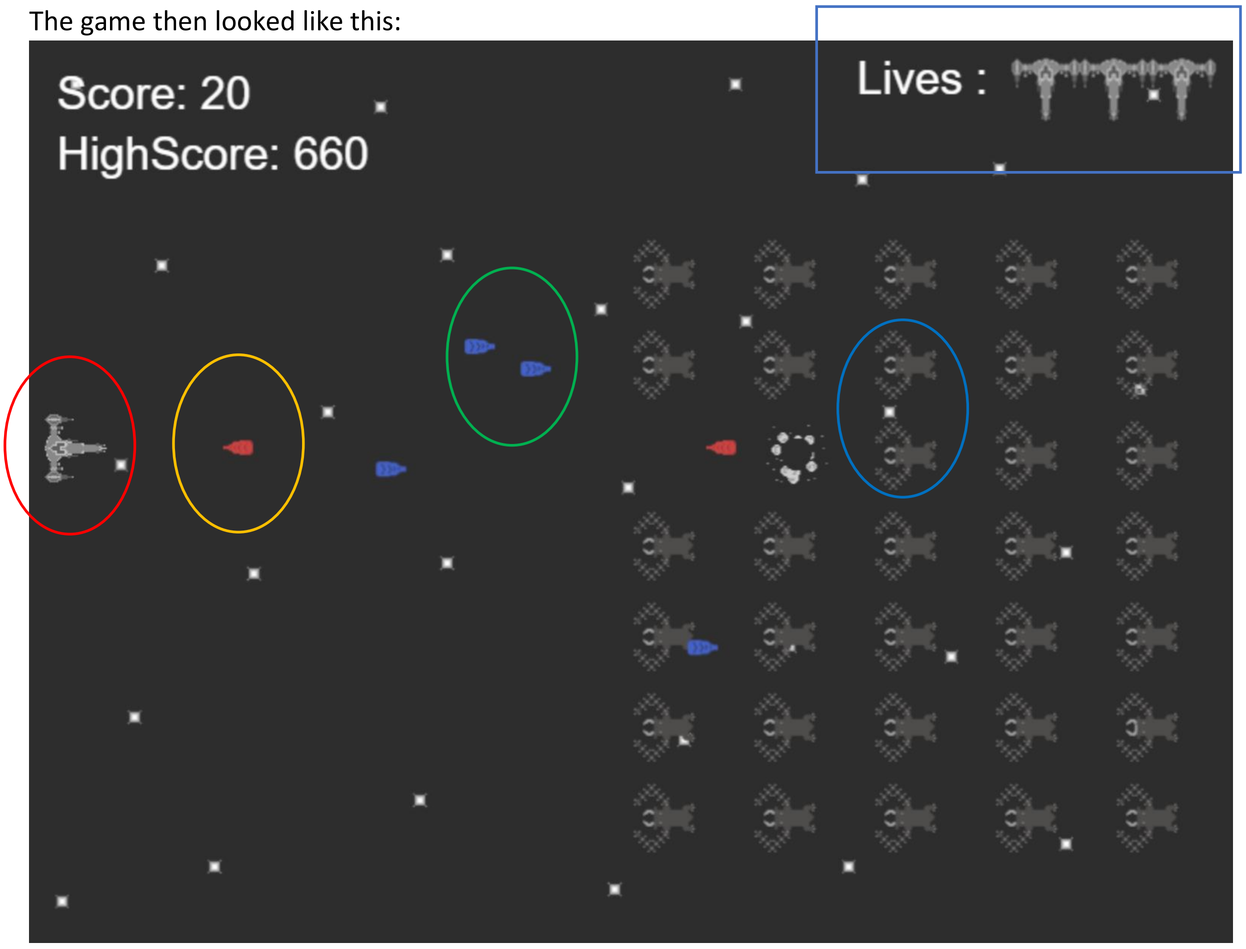
I then created a loop that created 3 separate player lives and displayed them next to the text I created earlier.



```
this.live = this.lives.getFirstAlive();

if (this.live)
{
    this.live.kill();

}


if (this.lives.countLiving() < 1)
{
    this.ship.kill();
    this.enemyBullets.callAll('kill');
    game.state.start('gameover');
}
,
```

Within the enemy hits player function I created a short piece of code that gets the first available player life and destroys it, if there are no player lives then it calls the game over state, which ends the game.

The game then looked like this:



Here you can see the player ship with some player projectiles flying towards some enemies that are also firing enemy projectiles. The player lives are displayed at the corner of the screen across from the player score and high score.

Problems / Insights

I had one major problem when I started developing my game and that was because I had lots of ideas that I thought might be possible to implement, without thinking about time constraints and my knowledge of the phaser library. This made it quite hard for me to start off my development process as I didn't really know where to start with the ideas I had within my head.

Other than that, I only had minor difficulties within implementing the sections into my game which was to do with certain items being misspelt or typos within the var / let names or certain calculations not adding up, which were easy to solve using "Console.log()" to track certain aspects.

Future Upgrades

Within the future I hope to add many components to this game, such as multiple stages that become increasingly difficult as the player goes on, Boss stages that would be different and have different tactics to defeat each one. A wider range of enemies and randomisation of them within the enemy formation. A Score board that could collect multiple players scores and contain them.

GitHub Link

https://github.com/Beando/GAD405-Phaser